April 2009

Motion Detection and Object Tracking with Interactive Surfaces

Jonathan Glumac
Worcester Polytechnic Institute

Project Number: FC CITZ

MOTION DETECTION AND OBJECT TRACKING WITH INTERACTIVE SURFACES

A Major Qualifying Project Report:

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

_____

Jonathan Glumac

Date: April 30th, 2009

_____

Professor Fabio Carrera, Project Advisor

_____

Professor Michael J. Ciaraldi, Project Co-Advisor

_____

Stephen Guerin, Project Sponsor

Abstract

I have developed software that allows users to interact with a surface using motion detection and object tracking. The system communicates user input to a projector, allowing the user to control the data being projected. The software will be integrated into a larger system to produce a completely interactive surface.

Table of Contents

1 Introduction 1
1.1 Current State 1
1.2 Project Focus 4
1.3 Definition of Terms 4
2 System Design 6
2.1 Project Requirements 6
2.2 System Setup 7
2.3 Design Decisions 7
2.3.1 Motion Detection Decisions 7
2.3.2 Object Tracking Decisions 9
2.3.3 Network Communication Decisions 11
3.1 Motion Detection 13
3.2 Object Tracking 15
3.3 Network Communication 18
4 Design Evaluation 19
4.1 Project Requirements 19
4.2 System Testing 20
4.3 Areas of Concern 20
5 Conclusion & Future Work 23
6. Bibliography 25

1 Introduction

The popularity of interactive screens and surfaces has created a large open source community for developing interactive media. A majority of these projects are artistic in nature, revolving around user interaction to control music, and color to create an artistic display of images and sounds. There are also some very practical applications for which these interactive surfaces can be used. An example would be an interactive map display that would allow for the interactive planning and presentation of city projects, like that in development for the Venice Project Center.

Professor Fabio Carrera introduced me to Stephen Guerin of Redfish Group, LLC located in Sante Fe, New Mexico. His company has been in collaboration with Professor Carrera to develop an interactive tabletop for use with the Venice Project Center. The interactive tabletop would be used for, although not limited to, city and canal maintenance planning and the presentation of different traffic management scenarios to city officials. The table would project a map that users around the table could interact with as they were discussing and planning city issues. This project is focused on the creation of the interactive input for the Venice table using motion detection and object tracking.

1.1 Current State

Currently there are several open source frameworks used for developing interactive surfaces. One of them is called touchLib developed by the NUI group. Another is the reacTIVision framework, developed at the Universitat Pompeu Fabra in Madrid Spain as part of the reacTable project. I will focus on the reacTIVision

framework and its implementation in the reacTable project, because it is very similar to the table in development at the Venice Project Center (Reactable, 2009).

The reacTable is a musical instrument that provides visual and audio feedback to coincide with the user's interactions. The surface is designed with the projector and the camera being mounted underneath the surface of the table as shown in *figure A.* A camera mounted under the table picks up any finger press made on the top of the table, and also has the ability to track certain objects with the use of fiducial markers. Fiducial markers are shapes that appear on an object that help define a tangible object. Examples of the fiducial symbols used by reacTable are shown in *figure B.*

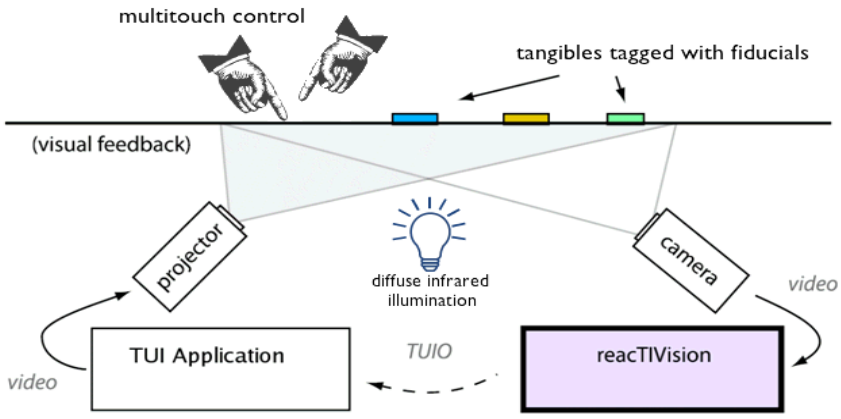


figure A (Reactable, 2009)

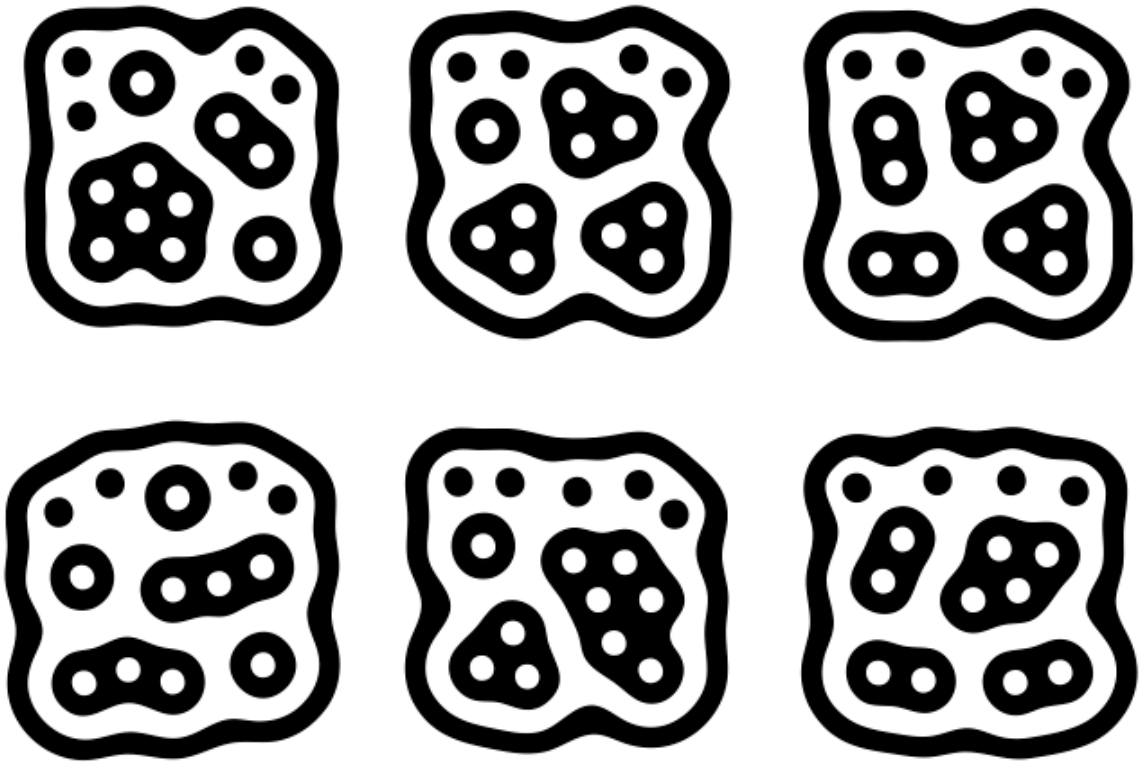


figure B (Reactable, 2009)

For the Venice table, the set up will be quite different. It will not require anything under the table surface and will instead have the camera and projector mounted above the table as seen in *Figure C*. The projector will display an image that the user can interact with. At the same time, the camera will be capturing the user and object interaction from above. The system will also have the ability to be integrated into other interactive surfaces that employ the same messaging protocol, such as the reacTable surface. This is meant to provide a new means of interaction with the surface as hands, fingers and objects are tracked from above.

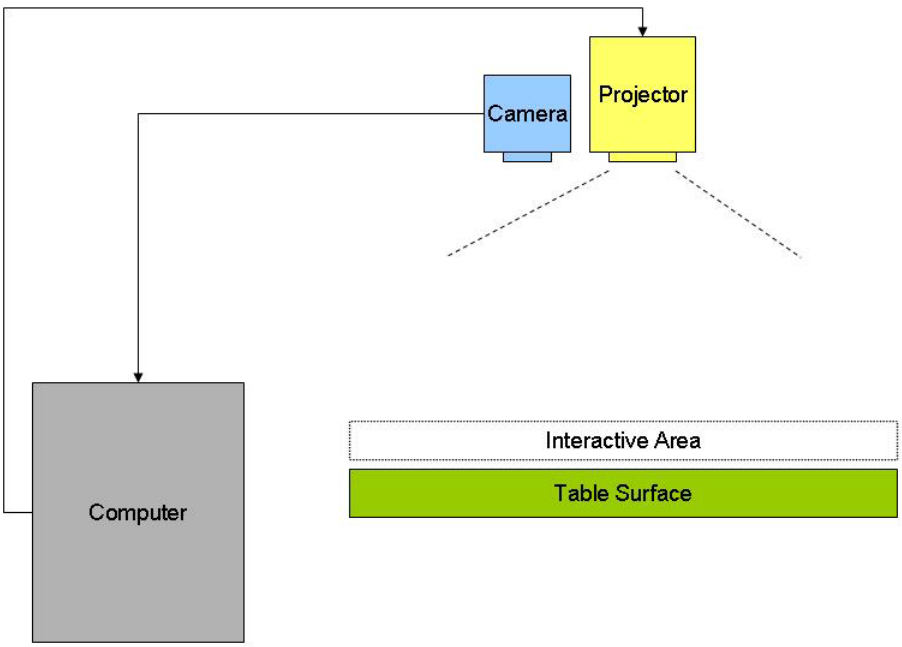


figure C

1.2 Project Focus

The focus of this project is to develop a system that can detect motion and track objects above or in front of a surface using a camera. The system will report to a client with all the information it has collected and processed. The product will be part of a larger system in which the client communicates with the projector to display new data. The projector will change or alter the image based on the interaction collected from the camera.

1.3 Definition of Terms

Blob: An area of detected movement in a frame.

Processing:	A java based visualization library.
Processing Sketch:	A Processing sketch is an open project in the Processing library.
OSC:	Stands for Open Sound Control, which is a network protocol.
TUIO:	A network protocol developed on top of the OSC protocol.

2 System Design

This section discusses the project requirements. Due to the proprietary nature of the project, I am limited in the amount of detail I can give regarding the design and source code.

2.1 Project Requirements

The project was proposed by Stephen Guerin with a number of requirements that needed to be fulfilled in order for the project to be a useful contribution to the Venice Interactive Table, as well as parameters that would ensure smooth integration.

The requirements were:

- The system must be able to detect objects in motion across the top of the surface.
- These objects must have a specific ID, which allows multiple objects to be tracked from frame to frame.
- The object tracking information must be reported through the Open Sound Control network protocol so that the data can be transferred amongst other interactive systems.
- An interactive example project must be implemented using the system that will be developed as a result of this project.

Along with these requirements, there was also some discussion of extra features that would be useful for the system. This included a finger detection algorithm, that would analyze the curvature of each blob and assess if the blob was a finger or not. This

would provide more information to the system which could be programmed to handle fingers and hands differently than other blobs.

2.2 System Setup

This system will only require a simple webcam setup on a computer. Other interactive surfaces might require different setups, but for the scope of this project the focus will stay on webcam motion detection and tracking. I used a Logitech 3000 webcam that has a USB connection.

2.3 Design Decisions

The project was split up into three modules, each concentrating on an important aspect of the overall project. They were divided into:

- Motion Detection
- Object Tracking
- Network Communication

2.3.1 Motion Detection Decisions

The motion detection module was dependant on how the object tracking was handled. This was not because motion detection was less important than object tracking, but rather because there were more options available for motion detection. Motion detection in general is handled in many different libraries, and has been utilized in several online examples. Object tracking however is more rare in libraries, and is not found in many applications other than the frameworks developed for these interactive surfaces.

Regardless, there were still some important design decisions to be made with respect to motion detection.

For the motion detection module we chose to implement the project using Processing. Processing is a Java based visualization language that is simple to learn, and would create the best opportunity for the development of user friendly features. OpenFrameworks was another library that was considered for the project. It is a C++ based visualization library. Processing was chosen because it already had much functionality we were looking for in this project.

The motion detection method that was selected is a background subtraction technique. A previous frame is used as the background, and its color is subtracted from the current frame to detect any differences in the new frame. This background frame will be resettable at any time by the user, but the first background frame used is the first frame that is processed by the application. A "sliding window" for the background frame was considered as well. The sliding time window would change the background frame at intervals so that it would come closer to what we are seeing in the current frames. This would improve the motion detection overall, but it would also eliminate any detected blobs that stop moving. Once the blob stops moving, the background sliding window would slowly incorporate the blobs into the background.

Optical flow was also considered as a means of motion detection, but its major weakness was that objects that stop moving are no longer detected. This would make it harder to track objects from screen to screen. With the background subtraction technique, we know the object is there even if its movement stops. Another consideration for motion detection was the Camshift algorithm. The only drawback of the Camshift

algorithm for this project was that it detects the distribution of a feature across the entire frame, such as color (Bradski & Kaehler, 2008). This application is meant to track any movement of any object, and using the Camshift algorithm would limit the objects the system was detecting. Background subtraction has its limitations as well, but there are image processing techniques which can limit "false positive" pixels in the frame (Bradski & Kaehler, 2008).

To help define the blobs I used the v3ga blob detection library. The v3ga library, which was developed for Processing, creates blob objects for any areas in the frame with a brightness level above a certain threshold. The thresholding of the v3ga library, coupled with the background subtraction technique mentioned above will detect areas of brightness in the differenced frame. This means the blob detection works best in areas of contrast between the current frame and the background frame. The routine will work on dark surfaces as well as light surfaces, but it will only be able to detect objects that have enough contrast compared to the background.

2.3.2 Object Tracking Decisions

Next we needed to figure out how to keep track of objects in motion from frame to frame. The v3ga stores our detected objects as blobs, which gives us a structure to track, but the library does not handle tracking the blobs from frame to frame. With modifications to the v3ga source code I implemented methods in the blobDetection class to keep track of the blobs that are being detected.

Object tracking was a particularly hard topic to research. Some libraries claiming to feature object tracking were only actually detecting the objects in motion and not

tracking their IDs from frame to frame. Similarly, computer vision books that handled the topic of object tracking were actually only discussing methods of motion detection.

There were two sources already available that did handle object tracking. The first was existing interactive surface frameworks called touchLib and reacTIVision. These were specifically developed for interactive surfaces. They both detected touch events on the table surface using infrared light, and then tracked the finger tip's motion. Either library would have been usable for my project if the blob tracking classes were not so heavily dependant on the rest of the library source. To use the blob tracking class required a full setup and configuration of an interactive tabletop. Even then I am not sure I would have been able to effectively use the class for our purposes. Also, using a tabletop-dependant library would complicate the possibility of a tabletop that had the camera and projector mounted above the table.

The next source that handled object tracking was part of the OpenCV library. It was a video surveillance pipeline that was developed to take care of motion detection with several available algorithms, as well as handling any object tracking. At first glance, the pipeline was the answer to the motion detection and object tracking requirements of the project. However, the pipeline is very cumbersome. It is not well documented, and is difficult to use. Parts of the pipeline are very processor heavy as well, consuming up to a gigabyte of RAM (OpenCV Wiki, 2009). I attempted to isolate the tracking algorithm used by the pipeline for use with my own detection routines. The lack of documentation and the reliance of the tracking module on the rest of the pipeline made it too difficult.

It was decided that implementing an object tracking algorithm would be the best approach. The v3ga blob detection library provided a blob structure on which I could base my tracking algorithm.

2.3.3 Network Communication Decisions

As a requirement, the network protocol selected for use with the project was the Open Sound Control (OSC). It was chosen because it is flexible and takes advantage of the User Datagram Protocol (UDP). It is also a popular choice among ongoing interactive projects, and can easily be imported into any Processing sketch. This allows for the quick integration into several projects.

We decided to implement the TUIO protocol within the system. The TUIO is a framework that defines a protocol for the transmission of OSC messages for multitouch surfaces. It provides a method for defining events, and bundling OSC messages together. This is the protocol used by the reacTable project, as well as some of the other projects being developed at Redfish Group in Sante Fe.

The TUIO protocol utilizes three types of OSC messages to communicate with the client application. They are alive, set and fseq messages.

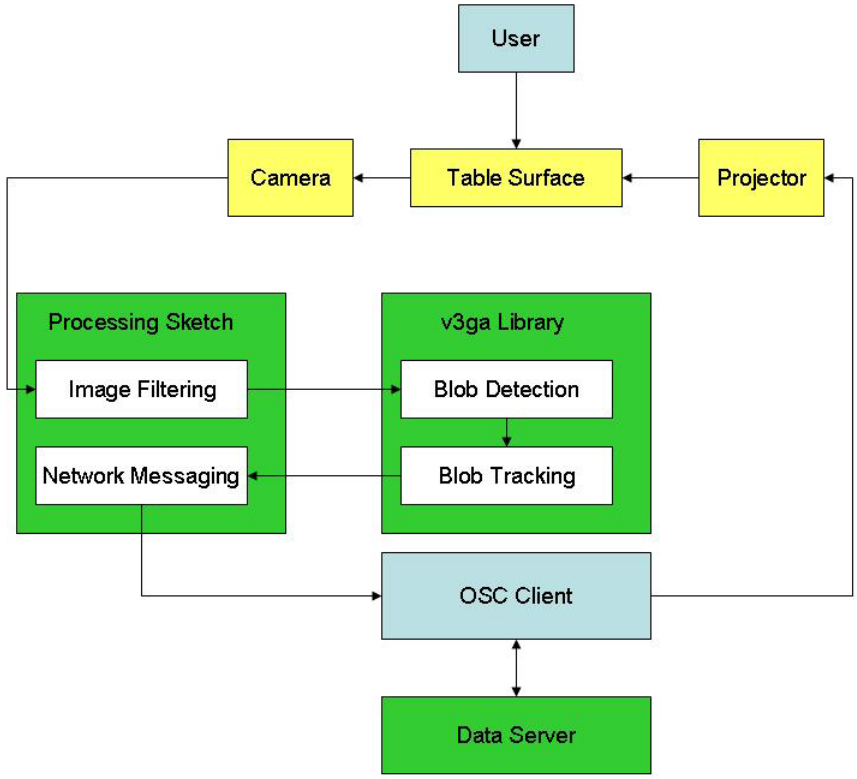
```
/tuio/[profileName] set [sessionID parameterList]
/tuio/[profileName] alive [list of active sessionIDs]
/tuio/[profileName] fseq [int32]
```

The protocol is designed to send packets or bundles of packets without listening for any return packets. This is because the system does not care about lost packets. If the client does not receive a packet, it will still get the information it needs in the following packets (TUIO, 2009). This makes the protocol ideal for our application, because we do not necessarily need all objects to be updated correctly every time. If a packet is lost for

one object, the necessary information will be sent in the next bundle in a set, or alive message.

3 Implementation

This section discusses the integration and development of the software.



3.1 Motion Detection

The motion detection was implemented in Processing. After opening up a video capture, each frame is fed into the detection phase. This consists of converting the video capture to an image, and then performing image enhancements.

First a background subtraction is performed on the frame. The color from each pixel in the background frame is subtracted from the color of each pixel in the current frame. The resulting image is brighter in areas of contrast, which should depict new foreground objects. The brightness comes from the absolute value of the frame

differencing. The closer the new pixel value is to 255, the closer the pixel color is to white. After background subtraction, image enhancements are performed using the Processing library's image filters. The image is filtered to eliminate noise, and to develop more contrast in the areas of interest.

```
img.filter(GRAY);
img.filter(THRESHOLD, .05f);
img.filter(ERODE);
img.filter(ERODE);
img.filter(DILATE);
img.filter(DILATE);
img.filter(BLUR, 3);
```

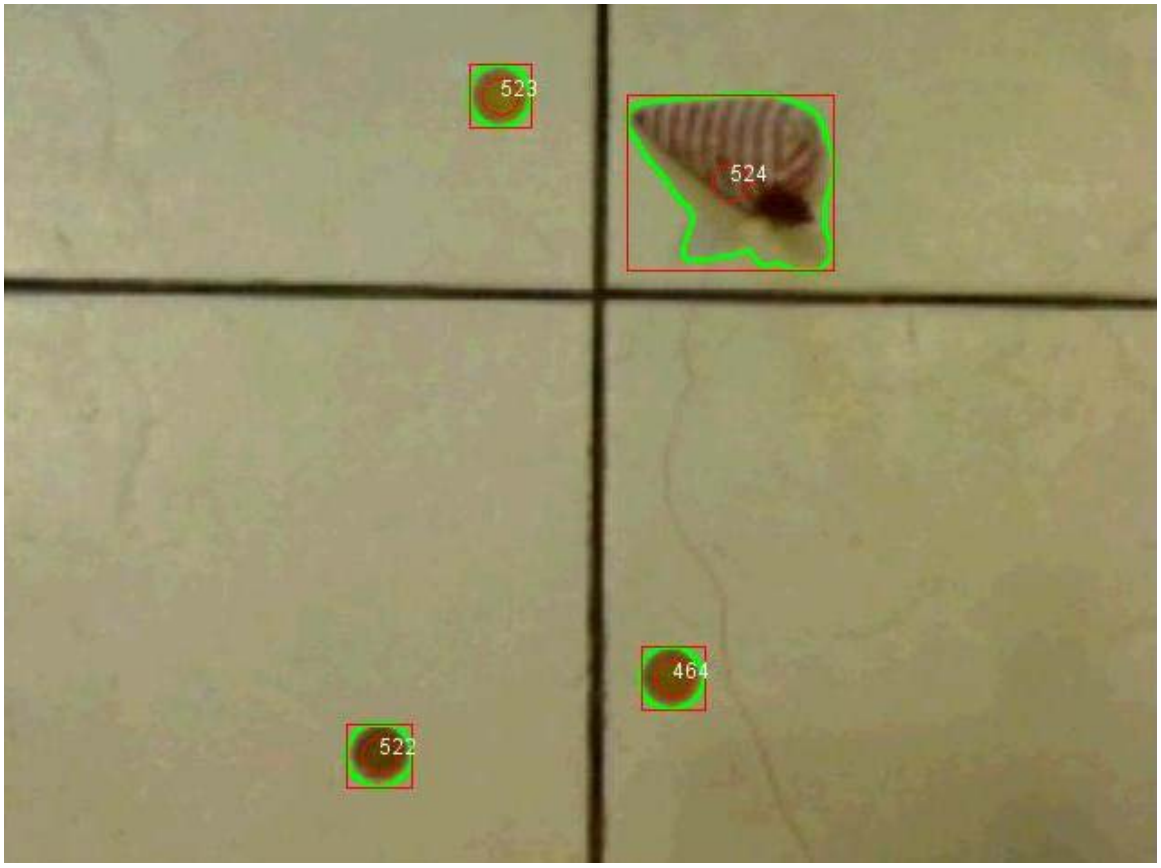
The image is run through a grayscale filter, and then an initial threshold filter. This is followed by erosion and dilation filtering. Erosion is used to eliminate the smaller areas of pixels and dilation to reestablish the remaining areas of pixels. This set up was detecting blobs, but was somewhat erratic. The edges of the blobs would jump around between frames. To settle the blobs down and make them more consistent a blur filter was added to the code, which performs a Gaussian blur on the image (Processing, 2009). The blur filter reduces noise and detail in the image which creates better blobs for the system to process.

Once the image has been enhanced it is sent through the v3ga blob detection routine. The blob detection routine uses a threshold for pixel brightness to determine where the blobs are on the screen. The threshold is adjustable within the processing sketch. All the detected blobs are then placed in a blob array.

```
theBlobDetection = new BlobDetection(img.width, img.height);
theBlobDetection.setThreshold(0.3f);
theBlobDetection.computeBlobs(img.pixels);
```

The library detects all blobs, and produces an array of blobs. A blob consists of an ID, a center coordinate, a height, a width and a set of vertices.

The detection library picks up even the smallest blobs and often some false positives that are not of interest to the system. It is left up to the Processing sketch to filter out blobs by size. Since the v3ga library was already being modified to handle object tracking, I also decided to implement a size limit on the blobs being detected. This will help cut down on the size of the detected blob array, and the amount of computation when comparing detected blobs to tracked blobs in the tracking module. In the frame below the system has detected the blobs seen outlined in green.

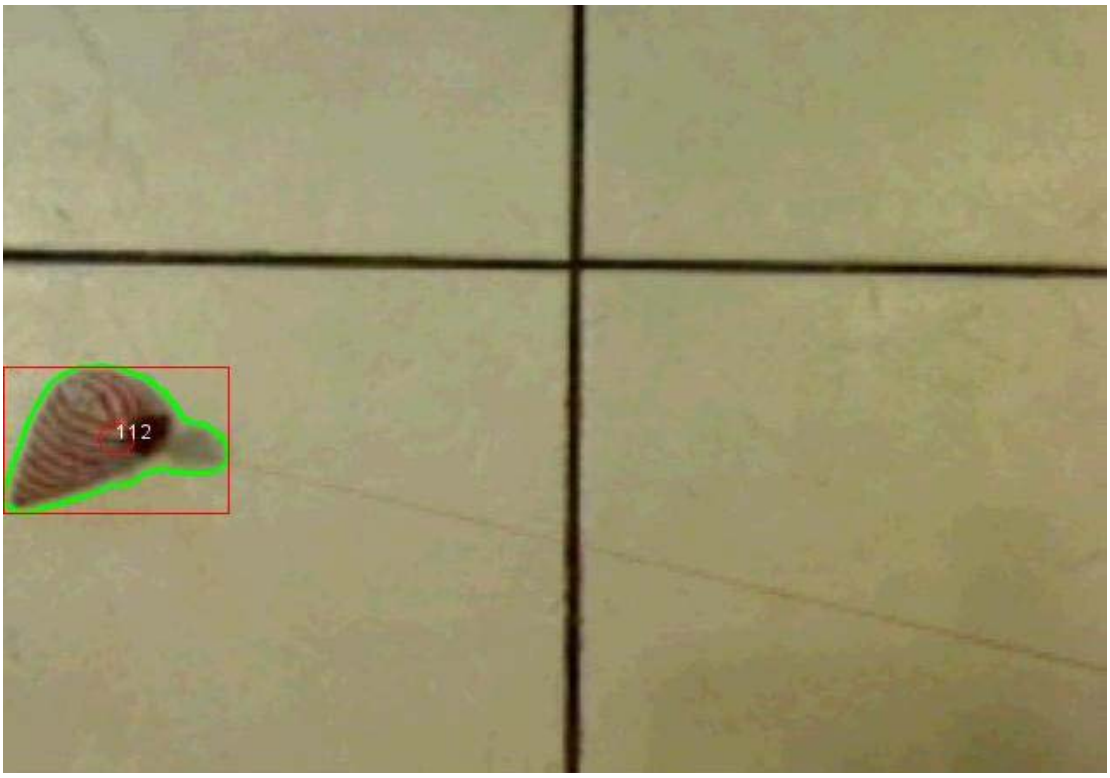


Detection Frame 1

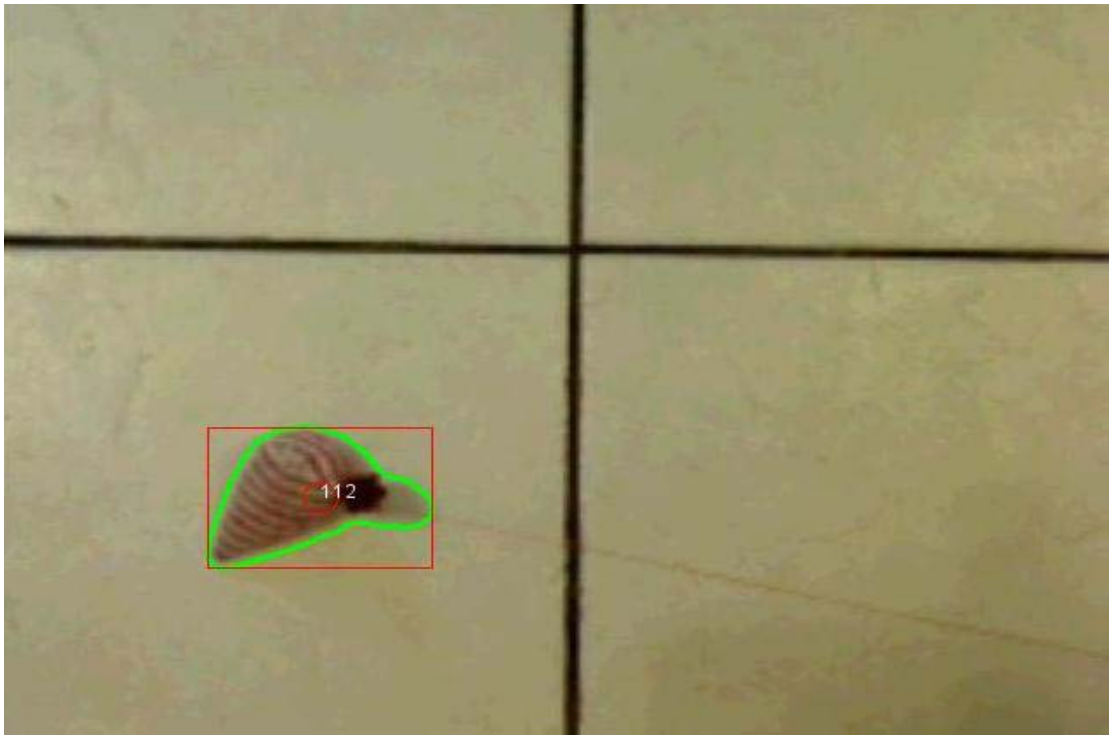
3.2 Object Tracking

The object tracking is handled within the blobDetection class of the v3ga library. As each frame is run through the blobDetection it collects a new array of detected blobs.

Whenever a call is made to the blobTracker method, these new blobs are compared to an array list of existing blobs from the previous frames that are considered to be tracked. The sequence of frames below displays the contours and the ID of an object as it is pulled across the surface by a thin thread.



Tracking Frame 1



Tracking Frame 2

Each blob that has been detected in the incoming frame is compared to the list of blobs that have been tracked from the previous frame. The x and y coordinates of the new blob's center is first checked to see if it is within a set boundary of the previous blob's center. If there is no match found, the new blob is given a special ID and is added to the tracked blob list. If there is exactly one match found between the new blob and the tracked list, then the match in the tracked list is updated with the data from the new blob. If more than one match is found, the blobs are then compared by size and height to determine which tracked blob is the better match for the new blob.

Comparisons of area, height and width are done to determine which blob is the best match for the incoming blob. The area of the new blob is compared to the area of each potential match. If this is determined not to be compelling enough, further comparisons between the height and width of each blob are done. Once a match has been

determined, the tracked blob is updated with the new blob data. Afterwards, all blobs from the tracked list that did not find a new blob match are deleted from the tracked list.

3.3 Network Communication

For the network communication we decided to implement the TUIO protocol. After the objects in the frame have been tracked, the processing sketch creates a *set* message for each tracked blob. The *set* message is used to tell the client that either a new object is on the screen, or to send updated object information for those objects already known by the client. The *set* messages are bundled together with an *alive* message which is used by the client to eliminate those objects it knows, that are no longer in the frame sequence.

4 Design Evaluation

This section discusses the testing and evaluation of the system that has been implemented.

4.1 Project Requirements

The requirements set forth at the beginning of the project were:

- The system must be able to detect objects in motion across the front of the screen.
- These objects must have a specific ID, which allows multiple objects to be tracked from frame to frame.
- The object tracking information must be reported through the OSC network protocol so that the data can be transferred amongst other interactive systems.
- An interactive example project must be implemented using the system that will be developed as a result of this project.

Each requirement has been met by the project. The system has the ability to detect objects in motion using a webcam. These objects are stored as blobs, which are then tracked by the methods implemented in the v3ga library. The OSC protocol is used to deliver blob information over a network or internet connection.

4.2 System Testing

The system was tested using ad hoc tests. I focused on the areas of the system that were most important, as well as spending time testing areas of concern. For all the tests, the camera was set three and six feet above a surface. The camera used for testing was the Logitech 3000 webcam. The webcam has a resolution of 640 x 480.

First the motion detection was tested by varying the threshold level between .3, .5, and .7 against different colored backgrounds. To test the motion tracking functionality, I used my hands as well as a laser pointer to ensure the system was following and naming the blobs properly. To test the collision of blobs and blob matching I used my hands and objects that were placed on the surface. To ensure the blob matching routines were used correctly I set break points in the code while experimenting with the collision of objects. Overall it was a tough aspect to test, because frames are processed so fast that changes in a blob ID are hard to follow.

For the network messaging, I used a TUIO Processing client application. It is set to replicate a simulation of the reacTable. Since I am using the same TUIO protocol, it provided good feedback for the messages I was sending.

4.3 Areas of Concern

There are some weaknesses and vulnerabilities in the motion detection and object tracking aspects of the project. There seem to be some problems with having several objects close to the camera lens. Testing determined that the camera should be at least three feet away from the objects it is trying to track. Performance is improved even more when objects are at a distance of six feet or greater. If an object is too close it has an

effect on the lighting that is entering the camera lens, which in turn has a drastic effect on the objects being tracked on the screen. This results in the appearance of several differences, even if there is no movement in those areas. In some cases it picks up several blobs that are not actually new objects. Also, objects that can span the entire frame can confuse the detection algorithm into thinking that the spaces around the object are blobs when in fact they are not. This can be limited by making sure that the objects are the minimum distance away from the camera. Adjustments to blob detection size may need to be made. Objects that are farther away from the lens take up fewer pixels in the frame.

There are also weaknesses that exist in the current tracking module. The size of the bounding box used when comparing new blobs with the tracked blobs can have an effect on performance. If the box is too small, then it is easy for objects moving quickly to lose their ID from frame to frame as they exit the boundary box before the next frame. This would lead to the loss of the old tracked blob and the creation of a new tracked blob for the same object from frame to frame. At the same time, the bounding box should not be too big because it could possibly encompass too many new blobs. This would lead to unnecessary comparison of blobs. The bounding box is adjustable by the user, but the issue exists for any size bounding box.

Another weakness involves the crossing and collision of blobs. When two blobs come into contact in a frame, it can combine them to create one larger blob. This would eliminate one of the blobs from existence in the tracking list. This poses a problem for situations involving object movement by the user. The hand of the user would almost

certainly corrupt the tracking of an object when placing it on the table or moving it to another spot on the table.

Some of the short comings in the system performance may be associated with the camera quality. For the implementation and testing of the system, I used the Logitech 3000 webcam. This is not the best camera, or even the best webcam. This could have led to the volatility of lighting, and focus in certain system setups. A high quality camera could improve the overall system.

5 Conclusion & Future Work

Overall, the design and implementation of this project is satisfactory for the needs of this project. The system has the ability to track objects on a surface, as well as any motion occurring around the surface. It is not known how well it will integrate into the overall system, but because of its simple design it should work well for its intended purpose. Because it is still a relatively simple implementation, after the system has been integrated with a table setup, each aspect of the system can be enhanced to improve its performance.

Major improvements can be made in the blob tracking module. A useful addition would be a tracking algorithm that was able to track blobs through collisions and the eclipsing of blobs. This would eliminate one major drawback of the current system, and improve interaction for the user. The exclusion of objects' shadows would be another great improvement for the system. Right now, objects can cast a shadow in certain lighting situations. This shadow is often picked up as part of the blob. To be able to eliminate the shadow from the blob detection would make the system much more accurate with its objects.

Along with improvements on existing features, there are also some features that could be introduced to the system. The ability to track fiducial markers would make the blob detection and tracking more dynamic.

Along with the idea of the interactive Venice table, Stephen is also investigating the use of this application in the development of interactive rooms. A possible scenario

would distribute several of the camera/projector couples throughout a room. In this room each user could be interacting with walls, the floor, each other and other surfaces.

6. Bibliography

Bradski, G.R., & Kaehler, A. (2008) Learning OpenCV: Computer Vision with the OpenCV Library. O'Rielly.

Davies, E.R. (2005) Machine Vision: Theory, Algorithms, Practicalities. Morgan Kaufmann.

TUIO Community (2009), http://www.tuio.org/ Pompeu Fabra University.

Reactable (2009), http://www.reactable.com/ Pompeu Fabra University.

OpenCV Wiki (2009), http://opencv.willowgarage.com/wiki/

Processing 1.0 (2009), http://processing.org/ Ben Fry and Casey Raes

Blob Detection (2009), http://www.v3ga.net/processing/BlobDetection/ v3ga.

oscP5 (2009), http://www.sojamo.de/libraries/oscP5/ Andreas Schlegel.